Reductions

Lecture 21
November 9, 2016
Part I

Reductions
Reductions

A reduction from Problem \textbf{X} to Problem \textbf{Y} means (informally) that if we have an algorithm for Problem \textbf{Y}, we can use it to find an algorithm for Problem \textbf{X}.

Using Reductions

1. We use reductions to find algorithms to solve problems.
2. We also use reductions to show that we \textit{can’t} find algorithms for some problems. (We say that these problems are \textit{hard}.)
## Example 1: Bipartite Matching and Flows

### How do we solve the **Bipartite Matching** Problem?

Given a bipartite graph $G = (U \cup V, E)$ and number $k$, does $G$ have a matching of size $\geq k$?

### Solution

Reduce it to **Max-Flow**. $G$ has a matching of size $\geq k$ iff there is a flow from $s$ to $t$ of value $\geq k$ in the auxiliary graph $G'$. 
Types of Problems

Decision, Search, and Optimization

1. **Decision problem.** Example: given $n$, is $n$ prime?

2. **Search problem.** Example: given $n$, find a factor of $n$ if it exists.

3. **Optimization problem.** Example: find the smallest prime factor of $n$. 
Optimization and Decision problems

For max flow...

Problem (Max-Flow optimization version)

Given an instance $G$ of network flow, find the maximum flow between $s$ and $t$.

Problem (Max-Flow decision version)

Given an instance $G$ of network flow and a parameter $K$, is there a flow in $G$, from $s$ to $t$, of value at least $K$?

While using reductions and comparing problems, we typically work with the decision versions. Decision problems have Yes/No answers. This makes them easy to work with.
A problem $\Pi$ consists of an infinite collection of inputs $\{I_1, I_2, \ldots, \}$. Each input is referred to as an instance.

The size of an instance $I$ is the number of bits in its representation.

For an instance $I$, $\text{sol}(I)$ is a set of feasible solutions to $I$.

For optimization problems each solution $s \in \text{sol}(I)$ has an associated value.
Examples

Example
An instance of **Bipartite Matching** is a bipartite graph, and an integer $k$. The solution to this instance is “YES” if the graph has a matching of size $\geq k$, and “NO” otherwise.

Example
An instance of **Max-Flow** is a graph $G$ with edge-capacities, two vertices $s, t$, and an integer $k$. The solution to this instance is “YES” if there is a flow from $s$ to $t$ of value $\geq k$, else ‘NO’.

What is an algorithm for a decision Problem $X$?
It takes as input an instance of $X$, and outputs either “YES” or “NO”.
Using reductions to solve problems

1. $\mathcal{R}$: Reduction $X \rightarrow Y$
2. $A_Y$: algorithm for $Y$:
3. $\Rightarrow$ New algorithm for $X$:

$$A_X(I_X):$$

// $I_X$: instance of $X$.
$I_Y \leftarrow \mathcal{R}(I_X)$
return $A_Y(I_Y)$

If $\mathcal{R}$ and $A_Y$ polynomial-time $\Rightarrow A_X$ polynomial-time.
Comparing Problems

1. “Problem $X$ is no harder to solve than Problem $Y$”.
2. If Problem $X$ reduces to Problem $Y$ (we write $X \leq Y$), then $X$ cannot be harder to solve than $Y$.
4. Equivalently, Max-Flow is at least as hard as Bipartite Matching.
5. $X \leq Y$:
   1. $X$ is no harder than $Y$, or
   2. $Y$ is at least as hard as $X$. 
Polynomial-time reductions

We say that an algorithm is **efficient** if it runs in polynomial-time.

To find efficient algorithms for problems, we are only interested in polynomial-time reductions. Reductions that take longer are not useful.

If we have a polynomial-time reduction from problem \( X \) to problem \( Y \) (we write \( X \leq_P Y \)), and a poly-time algorithm \( A_Y \) for \( Y \), we have a polynomial-time/efficient algorithm for \( X \).

\[
\begin{array}{c}
I_X \xrightarrow{\mathcal{R}} I_Y \xrightarrow{A_Y} \text{YES} \\
A_X \xrightarrow{\mathcal{R}} I_Y \xrightarrow{A_Y} \text{NO}
\end{array}
\]
A polynomial time reduction from a decision problem $X$ to a decision problem $Y$ is an algorithm $\mathcal{A}$ that has the following properties:

1. given an instance $I_X$ of $X$, $\mathcal{A}$ produces an instance $I_Y$ of $Y$
2. $\mathcal{A}$ runs in time polynomial in $|I_X|$.
3. Answer to $I_X$ YES iff answer to $I_Y$ is YES.

**Proposition**

If $X \leq_P Y$ then a polynomial time algorithm for $Y$ implies a polynomial time algorithm for $X$.

Such a reduction is called a **Karp reduction**. Most reductions we will need are Karp reductions.
Let $X$ and $Y$ be two decision problems, such that $X$ can be solved in polynomial time, and $X \leq_p Y$. Then

(A) $Y$ can be solved in polynomial time.
(B) $Y$ can NOT be solved in polynomial time.
(C) If $Y$ is hard then $X$ is also hard.
(D) None of the above.
(E) All of the above.
Polynomial-time reductions and hardness

For decision problems $X$ and $Y$, if $X \leq_P Y$, and $Y$ has an efficient algorithm, $X$ has an efficient algorithm.

If you believe that Independent Set does not have an efficient algorithm, why should you believe the same of Clique?

Because we showed Independent Set $\leq_P$ Clique. If Clique had an efficient algorithm, so would Independent Set!

If $X \leq_P Y$ and $X$ does not have an efficient algorithm, $Y$ cannot have an efficient algorithm!
Proposition

Let $\mathcal{R}$ be a polynomial-time reduction from $X$ to $Y$. Then for any instance $I_X$ of $X$, the size of the instance $I_Y$ of $Y$ produced from $I_X$ by $\mathcal{R}$ is polynomial in the size of $I_X$.

Proof.

$\mathcal{R}$ is a polynomial-time algorithm and hence on input $I_X$ of size $|I_X|$ it runs in time $p(|I_X|)$ for some polynomial $p()$. $I_Y$ is the output of $\mathcal{R}$ on input $I_X$. $\mathcal{R}$ can write at most $p(|I_X|)$ bits and hence $|I_Y| \leq p(|I_X|)$.  

Note: Converse is not true. A reduction need not be polynomial-time even if output of reduction is of size polynomial in its input.
Polynomial-time Reduction

A polynomial time reduction from a *decision* problem $\mathbf{X}$ to a *decision* problem $\mathbf{Y}$ is an *algorithm* $\mathcal{A}$ that has the following properties:

1. Given an instance $\mathbf{I}_X$ of $\mathbf{X}$, $\mathcal{A}$ produces an instance $\mathbf{I}_Y$ of $\mathbf{Y}$.
2. $\mathcal{A}$ runs in time polynomial in $|\mathbf{I}_X|$. This implies that $|\mathbf{I}_Y|$ (size of $\mathbf{I}_Y$) is polynomial in $|\mathbf{I}_X|$.
3. Answer to $\mathbf{I}_X$ YES *iff* answer to $\mathbf{I}_Y$ is YES.

**Proposition**

If $\mathbf{X} \leq_p \mathbf{Y}$ then a polynomial time algorithm for $\mathbf{Y}$ implies a polynomial time algorithm for $\mathbf{X}$.

Such a reduction is called a Karp reduction. Most reductions we will need are Karp reductions.
Transitivity of Reductions

**Proposition**

\[ X \leq_P Y \text{ and } Y \leq_P Z \text{ implies that } X \leq_P Z. \]

Note: \( X \leq_P Y \) does not imply that \( Y \leq_P X \) and hence it is very important to know the FROM and TO in a reduction.

To prove \( X \leq_P Y \) you need to show a reduction FROM \( X \) TO \( Y \) in other words show that an algorithm for \( Y \) implies an algorithm for \( X \).
Using Reductions to show Hardness

Here, we say that a problem is “hard” if there is no polynomial-time algorithm known for it (and it is believed that such an algorithm does not exist)

- Start with an existing “hard” problem $X$
- Prove that $X \leq_P Y$
- Then we have shown that $Y$ is a “hard” problem
Examples of hard problems

Problems

1. SAT
2. 3SAT
3. Independent Set and Clique
4. Vertex Cover
5. Set Cover
6. Hamilton Cycle
7. Knapsack and Subset Sum and Partition
8. Integer Programming
9. ...
Part II

Examples of Reductions
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
2. **clique**: every pair of vertices in $V'$ is connected by an edge of $G$. 
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
2. **clique**: every pair of vertices in $V'$ is connected by an edge of $G$. 

![Graph Diagram]
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
2. **clique**: every pair of vertices in $V'$ is connected by an edge of $G$. 

![Diagram of a graph with red and white vertices]

- Red vertices represent the independent set.
- White vertices represent the clique.

Chandra & Ruta (UIUC) 
CS473 
Fall 2016
Independent Sets and Cliques

Given a graph $G$, a set of vertices $V'$ is:

1. **independent set**: no two vertices of $V'$ connected by an edge.
2. **clique**: every pair of vertices in $V'$ is connected by an edge of $G$. 
The **Independent Set** and **Clique** Problems

**Problem: Independent Set**

**Instance:** A graph $G$ and an integer $k$.

**Question:** Does $G$ has an independent set of size $\geq k$?

**Problem: Clique**

**Instance:** A graph $G$ and an integer $k$.

**Question:** Does $G$ has a clique of size $\geq k$?
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$. 
An instance of **Independent Set** is a graph $G$ and an integer $k$. 
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$. Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is **not** an edge of $G$. ($\overline{G}$ is the **complement** of $G$.) We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$.

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\overline{G}$ is the *complement* of $G$.)

We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$.

Convert $G$ to $\overline{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\overline{G}$ is the complement of $G$.) We use $\overline{G}$ and $k$ as the instance of **Clique**.
Reducing **Independent Set** to **Clique**

An instance of **Independent Set** is a graph $G$ and an integer $k$.

Convert $G$ to $\bar{G}$, in which $(u, v)$ is an edge iff $(u, v)$ is not an edge of $G$. ($\bar{G}$ is the *complement* of $G$.)

We use $\bar{G}$ and $k$ as the instance of **Clique**.
Independent Set and Clique

1. \textbf{Independent Set} $\leq$ \textbf{Clique}.
   What does this mean?

2. If have an algorithm for \textbf{Clique}, then we have an algorithm for \textbf{Independent Set}.

3. \textbf{Clique} is \textit{at least as hard as } \textbf{Independent Set}.

4. Also... \textbf{Independent Set} is \textit{at least as hard as } \textbf{Clique}.
Vertex Cover

Given a graph \( G = (V, E) \), a set of vertices \( S \) is:

1. A **vertex cover** if every \( e \in E \) has at least one endpoint in \( S \).
Vertex Cover

Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. A **vertex cover** if every $e \in E$ has at least one endpoint in $S$. 

![Graph Diagram]
Given a graph $G = (V, E)$, a set of vertices $S$ is:

- A **vertex cover** if every $e \in E$ has at least one endpoint in $S$. 

![Graph Example](https://via.placeholder.com/150)
Given a graph $G = (V, E)$, a set of vertices $S$ is:

1. A **vertex cover** if every $e \in E$ has at least one endpoint in $S$. 
The **Vertex Cover** Problem

**Problem (Vertex Cover)**

**Input:** A graph $G$ and integer $k$.

**Goal:** Is there a vertex cover of size $\leq k$ in $G$?

Can we relate **Independent Set** and **Vertex Cover**?
Relationship between...

Vertex Cover and Independent Set

Proposition

Let $G = (V, E)$ be a graph. $S$ is an independent set if and only if $V \setminus S$ is a vertex cover.

Proof.

$(\Rightarrow)$ Let $S$ be an independent set

1. Consider any edge $uv \in E$.
2. Since $S$ is an independent set, either $u \notin S$ or $v \notin S$.
3. Thus, either $u \in V \setminus S$ or $v \in V \setminus S$.
4. $V \setminus S$ is a vertex cover.

$(\Leftarrow)$ Let $V \setminus S$ be some vertex cover:

1. Consider $u, v \in S$
2. $uv$ is not an edge of $G$, as otherwise $V \setminus S$ does not cover $uv$.
3. $\implies S$ is thus an independent set.
**Independent Set $\leq_p$ Vertex Cover**

1. **G**: graph with $n$ vertices, and an integer $k$ be an instance of the **Independent Set** problem.

2. **G** has an independent set of size $\geq k$ iff **G** has a vertex cover of size $\leq n - k$.

3. $(G, k)$ is an instance of **Independent Set**, and $(G, n - k)$ is an instance of **Vertex Cover** with the same answer.

4. Therefore, **Independent Set $\leq_p$ Vertex Cover**. Also **Vertex Cover $\leq_p$ Independent Set**.
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?

**Example**

Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

\[
S_1 = \{3, 7\} \quad S_2 = \{3, 4, 5\} \\
S_3 = \{1\} \quad S_4 = \{2, 4\} \\
S_5 = \{5\} \quad S_6 = \{1, 2, 6, 7\}
\]
The **Set Cover** Problem

**Problem (Set Cover)**

**Input:** Given a set $U$ of $n$ elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$.

**Goal:** Is there a collection of at most $k$ of these sets $S_i$ whose union is equal to $U$?

**Example**

Let $U = \{1, 2, 3, 4, 5, 6, 7\}$, $k = 2$ with

- $S_1 = \{3, 7\}$
- $S_2 = \{3, 4, 5\}$
- $S_3 = \{1\}$
- $S_4 = \{2, 4\}$
- $S_5 = \{5\}$
- $S_6 = \{1, 2, 6, 7\}$

$\{S_2, S_6\}$ is a set cover
Vertex Cover $\leq_p$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:
Vertex Cover $\leq_P$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of $\text{Vertex Cover}$, construct an instance of $\text{Set Cover}$ as follows:

1. Number $k$ for the $\text{Set Cover}$ instance is the same as the number $k$ given for the $\text{Vertex Cover}$ instance.
Vertex Cover $\leq_P$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.
2. $U = E$. 
Vertex Cover $\leq^P$ Set Cover

Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.

2. $U = E$.

3. We will have one set corresponding to each vertex; $S_v = \{e \mid e$ is incident on $v\}$.
Given graph $G = (V, E)$ and integer $k$ as instance of Vertex Cover, construct an instance of Set Cover as follows:

1. Number $k$ for the Set Cover instance is the same as the number $k$ given for the Vertex Cover instance.

2. $U = E$.

3. We will have one set corresponding to each vertex; $S_v = \{e \mid e \text{ is incident on } v\}$.

Observe that $G$ has vertex cover of size $k$ if and only if $U, \{S_v\}_{v \in V}$ has a set cover of size $k$. (Exercise: Prove this.)
Vertex Cover $\leq_p$ Set Cover: Example

Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with $S_1 = \{c, g\}$, $S_2 = \{b, d\}$, $S_3 = \{c, d, e\}$, $S_4 = \{e, f\}$, $S_5 = \{a\}$, $S_6 = \{a, b, f, g\}$.

$\{S_3, S_6\}$ is a set cover.
Vertex Cover $\leq_P$ Set Cover: Example

Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with

- $S_1 = \{c, g\}$
- $S_2 = \{b, d\}$
- $S_3 = \{c, d, e\}$
- $S_4 = \{e, f\}$
- $S_5 = \{a\}$
- $S_6 = \{a, b, f, g\}$

$\{S_3, S_6\}$ is a set cover.
Vertex Cover $\leq_P$ Set Cover: Example

Let $U = \{a, b, c, d, e, f, g\}$, $k = 2$ with

- $S_1 = \{c, g\}$
- $S_2 = \{b, d\}$
- $S_3 = \{c, d, e\}$
- $S_4 = \{e, f\}$
- $S_5 = \{a\}$
- $S_6 = \{a, b, f, g\}$

$\{S_3, S_6\}$ is a set cover

$\{3, 6\}$ is a vertex cover
Proving Reductions

To prove that $X \leq_P Y$ you need to give an algorithm $A$ that:

1. Transforms an instance $I_X$ of $X$ into an instance $I_Y$ of $Y$.
2. Satisfies the property that answer to $I_X$ is YES iff $I_Y$ is YES.
   
   1. typical easy direction to prove: answer to $I_Y$ is YES if answer to $I_X$ is YES
   2. typical difficult direction to prove: answer to $I_X$ is YES if answer to $I_Y$ is YES (equivalently answer to $I_X$ is NO if answer to $I_Y$ is NO).
3. Runs in polynomial time.
Example of incorrect reduction proof

Try proving \textbf{Matching} \leq \textbf{P} \textbf{Bipartite Matching} via following reduction:

1. Given graph \( G = (V, E) \) obtain a bipartite graph \( G' = (V', E') \) as follows.
   1. Let \( V_1 = \{u_1 \mid u \in V\} \) and \( V_2 = \{u_2 \mid u \in V\} \). We set \( V' = V_1 \cup V_2 \) (that is, we make two copies of \( V \))
   2. \( E' = \{u_1v_2 \mid u \neq v \text{ and } uv \in E\} \)
2. Given \( G \) and integer \( k \) the reduction outputs \( G' \) and \( k \).
Claim

Reduction is a poly-time algorithm. If $G$ has a matching of size $k$ then $G'$ has a matching of size $k$.

Proof.

Exercise.

Claim

If $G'$ has a matching of size $k$ then $G$ has a matching of size $k$. 

Incorrect! Why?

Vertex $u \in V$ has two copies $u_1$ and $u_2$ in $G'$. A matching in $G'$ may use both copies!
Claim

Reduction is a poly-time algorithm. If $G$ has a matching of size $k$ then $G'$ has a matching of size $k$.

Proof.

Exercise.

Claim

If $G'$ has a matching of size $k$ then $G$ has a matching of size $k$.

Incorrect! Why?
**Claim**

*Reduction is a poly-time algorithm. If \( G \) has a matching of size \( k \) then \( G' \) has a matching of size \( k \).*

**Proof.**

Exercise.

**Claim**

*If \( G' \) has a matching of size \( k \) then \( G \) has a matching of size \( k \).*

Incorrect! Why? Vertex \( u \in V \) has two copies \( u_1 \) and \( u_2 \) in \( G' \). A matching in \( G' \) may use both copies!
Problem: Subset Sum

**Instance:** \( S \) - set of positive integers, \( t \): - an integer number (target).

**Question:** Is there a subset \( X \subseteq S \) such that \( \sum_{x \in X} x = t \)?

Problem: Partition

**Instance:** A set \( S \) of \( n \) numbers.

**Question:** Is there a subset \( T \subseteq S \) s.t. \( \sum_{t \in T} t = \sum_{s \in S \setminus T} s \)?

Assume that we can solve **Subset Sum** in polynomial time, then we can solve **Partition** in polynomial time. This statement is

(A) True.

(B) Mostly true.

(C) False.

(D) Mostly false.
II: Partition and subset sum?

**Problem: Partition**

**Instance:** A set $S$ of $n$ numbers.

**Question:** Is there a subset $T \subseteq S$ s.t. $\sum_{t \in T} t = \sum_{s \in S \setminus T} s$?

**Problem: Subset Sum**

**Instance:** $S$ - set of positive integers, $t$: - an integer number (target).

**Question:** Is there a subset $X \subseteq S$ such that $\sum_{x \in X} x = t$?

Assume that we can solve **Partition** in polynomial time, then we can solve **Subset Sum** in polynomial time. This statement is

(A) True.

(B) Mostly true.

(C) False.

(D) Mostly false.